Cloud-Based Data Processing

Distributed Data: Partitioning

Jana Giceva

Replication

- What are the benefits of replication?
- What are the challenges?
- In leader-based replication, the leader sends a replication log to its followers. What are different ways to implement it?
- What must one be careful of when issuing the read requests to the followers?
- What is leaderless replication?
  - How is it implemented?
  - Give an example of a quorum.

Replication vs. Partitioning

- There are two common ways data is distributed across multiple nodes.

- **Replication**
  - Keeps a copy of the same data on different nodes (potentially different locations).
  - Provides redundancy – If some nodes are unavailable, others can continue serving requests.
  - Reduces latency especially for high load or wide distribution of users across the globe.

- **Partitioning**
  - Split the big dataset into smaller subsets called *partitions.*
  - Each partition placed on a separate node.

- One can combine both replication and partitioning!

Partitioning

Partitioning

- For very **large datasets**, or very **high throughput**, we need to break the data up into **partitions.**
- **Q: Why?**
- Clarifying terminology:
  - What we call a **partition** here is called a *shard* in MongoDB, Elasticsearch, and SolrCloud; *region* in Hbase, a *tablet* in BigTable, a *vnode* in Cassandra and Riak, and a *vBucket* in Couchbase.
- Partitions are defined in such a way that a piece of data belongs to **exactly one partition.**

Why partition data?

- **Improve scalability**
  - Different partitions can be placed on different nodes in a shared nothing cluster
- **Improve performance**
  - Data operations on each partition work on **smaller** data **volume**
  - Operations that affect more than one partition can run in **parallel**
- **Improve security**
  - Can separate sensitive and non-sensitive data into different partitions and apply different security controls to the sensitive data
- **Improve availability**
  - Avoid a single point of failure. If one partition becomes unavailable, the others are still intact.
- **Allows better customization**

Designing partitions

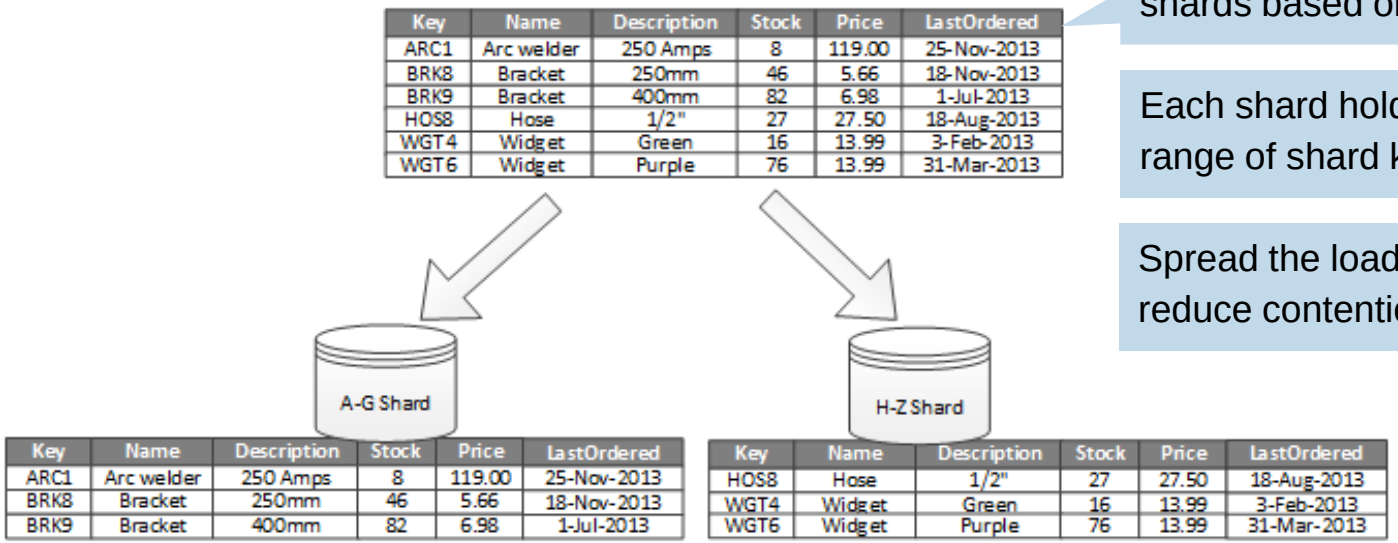
- **Horizontal partition (sharding):**
  - Each partition is a separate data store, but all partitions have the same schema
  - Each partition is known as a *shard* and holds a specific subset of the data
    - e.g., all the orders for a specific set of customers

- **Vertical partitioning:**
  - Each partition holds a *subset of the fields* for items in the data store
    - e.g., frequently accessed fields, may be placed in one vertical partition and less frequently accessed fields in another.

- **Functional partitioning:**
  - Data is aggregated according to how it is used by each bounded context in the system
    - e.g., An e-commerce might store *invoice data* in one partition and *product inventory data* in another

Horizontal partitioning (sharding)

- Example horizontal partitioning or sharding



Product inventory data is divided into shards based on the product key.

Each shard holds the data for a cont. range of shard keys (A-G and H-Z)

Spread the load over more nodes, to reduce contention and response time.

Horizontal partitioning (sharding)

- The most important factor is the **choice of sharding key.**

- **Q: What's should we optimize for with the sharding key?**

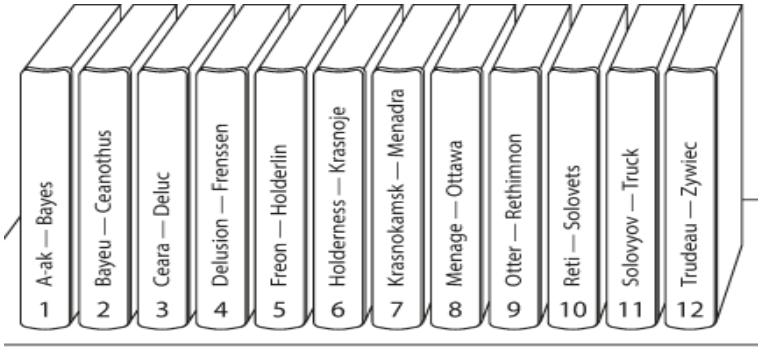
  - **Goal** is
    - Not necessarily to have the shards the same size, but
    - to **spread** the **data** and query **load evenly** across the nodes.

- **Q: What if the partitioning is not fair?**

  - If the partitioning is unfair, some partitions will have more data or queries, we call it **skewed.**
  - A partition with disproportionally high load is called **a hot spot.**

Horizontal Partitioning strategies

- **by Key Range**
  - Assign a **continuous range of keys** to each **partition**.
  - The range of keys are not necessarily evenly spaced, because your data may not be evenly distributed.
    BigTable, Hbase, RethinkDB, and MongoDB before v2.4

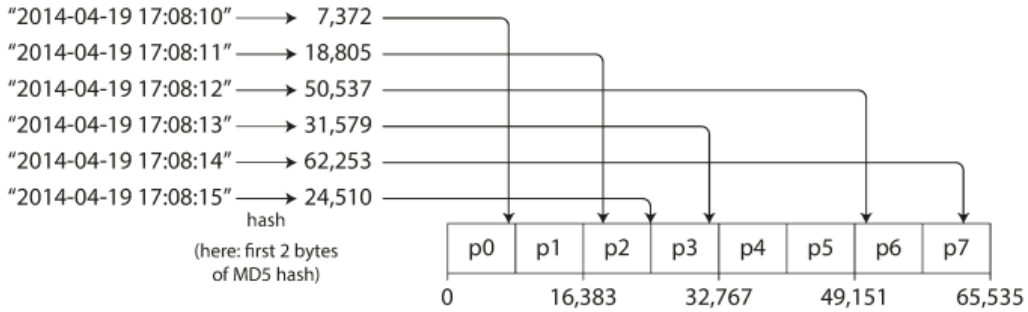


- **Advantage:**
  - Within each partition we can keep the keys in sorted order
    → range scans are fast and easy
  - Can fetch several related records in one query

- **Disadvantage:**
  - Certain access patterns can lead to hot spots

Horizontal Partitioning strategies II

- **by Hash of Key**
  - **hash a key** to determine the partition
  - a **partition** for a **range of hashes**
  - if a key's hash value belongs to a partition's range then the key is placed in that partition.

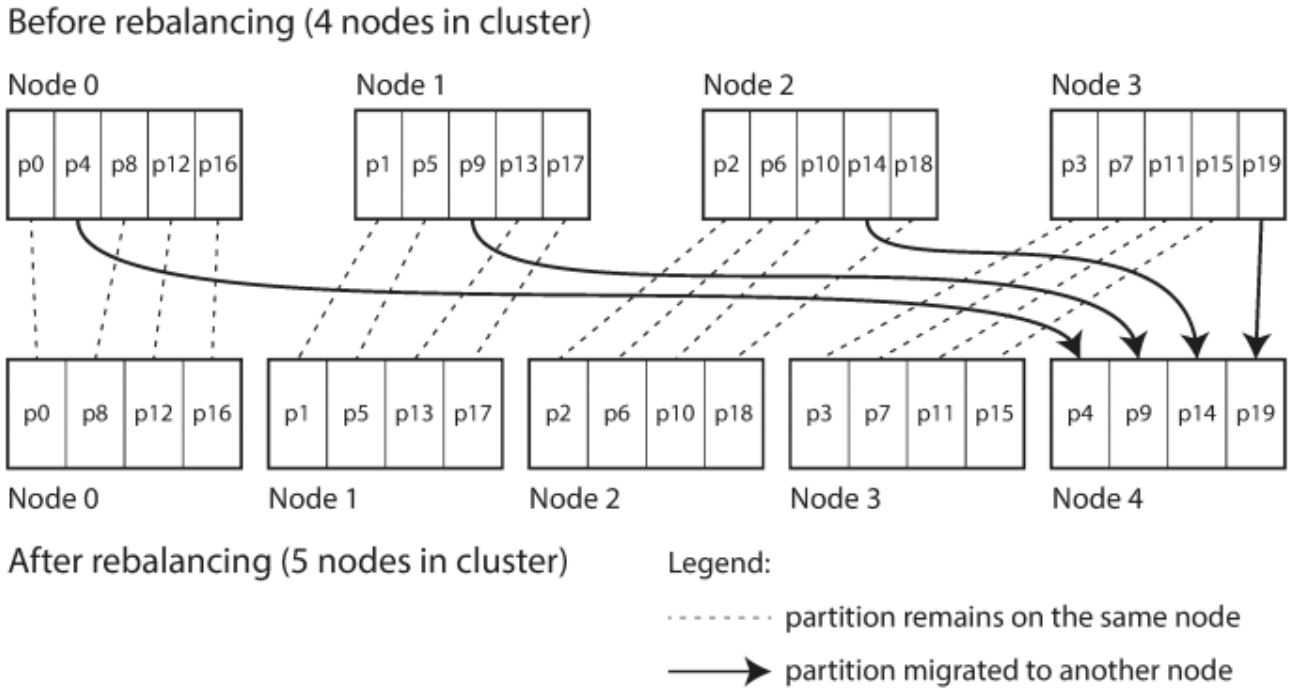


- **Advantage:**
  - No problem with skew and hot spots (overstatement, we may still have issues, but they are rare)

- **Disadvantage:**
  - No longer easy to do efficient range queries.
  - e.g., range queries on the primary key are not supported by Riak, Couchbase or Voldemort.

Rebalancing partitions

- **Rebalancing is often necessary**



- **Strategies of rebalancing:**
- **Q: How not to do it?**
  - **Hash mod N.**
  - If the number of nodes N changes, most of the keys will need to be moved from one node to another.

Rebalancing partitions

- **Rebalancing is often necessary**

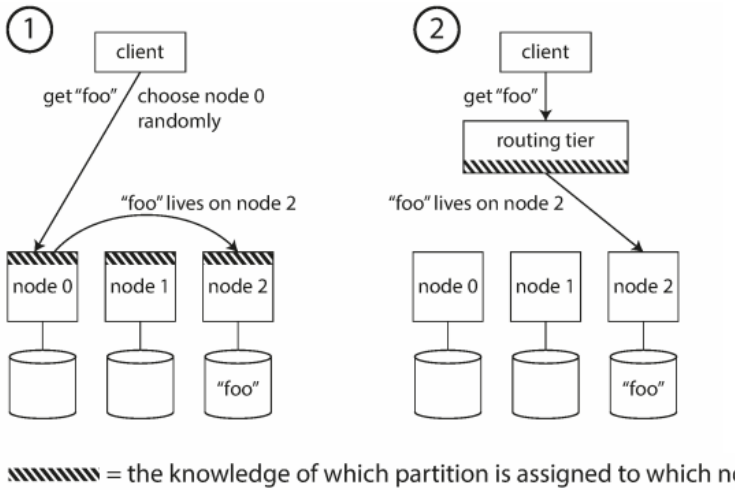
- **Strategies of rebalancing:**
- **Q: Can you think of a better way?**
  - **Fix the number of partitions P so that P >> N**
    - If a node is removed/added to the cluster,
      only a few (entire) partitions need to be moved.
    - The number of partitions remains the same, and the assignment of keys to partitions is not changed.

- **Q: What happens when a partition's size exceeds the limit?**
  - split it into two (like in a B-tree).
  - Dynamic partitioning
  - Applicable with range and hash partitioning

- **Q: How do you ensure proportional load across the nodes?**
  - Have a fixed number of partitions per node.

Request routing

- **Open question: when a client wants to make a request, how does it know which node to ask?**
  - As partitions are rebalanced, the assignment of partitions to nodes changes
  - Someone needs to have the top-level overview.



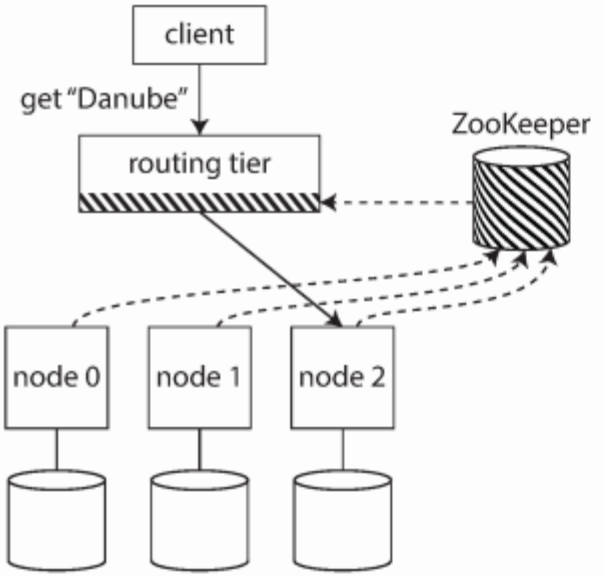
- **Three main options:**
  - The node layer
  - The routing tier (or third party)
  - The clients

  - It is a challenging problem as all participants need to agree → requires reaching a consensus.

- Many systems rely on a **coordination service** such as Zookeeper to keep track of cluster meta data.
- Others use alternatives like **gossip protocol** among the nodes to disseminate cluster state changes.

Example using ZooKeeper to keep track

- The routing tier can subscribe to this information from the ZooKeeper service



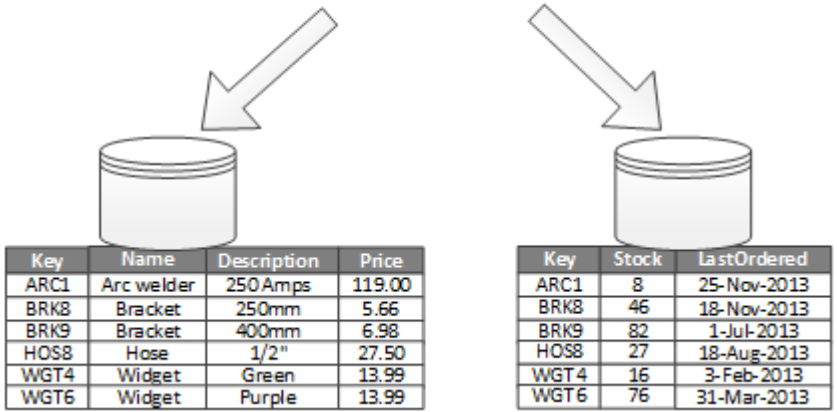
Key range	Partition	Node	IP address
A-ak — Bayes	partition 0	node 0	10.20.30.100
Bayeu — Ceanothus	partition 1	node 1	10.20.30.101
Ceara — Deluc	partition 2	node 2	10.20.30.102
Delusion — Frenssen	partition 3	node 0	10.20.30.100
Freon — Holderlin	partition 4	node 1	10.20.30.101
Holderness — Krasnoje	partition 5	node 2	10.20.30.102
Krasnokamsk — Menadra	partition 6	node 0	10.20.30.100
Menage — Ottawa	partition 7	node 1	10.20.30.101
Otter — Rethimnon	partition 8	node 2	10.20.30.102
Reti — Solovets	partition 9	node 0	10.20.30.100
Solovyov — Truck	partition 10	node 1	10.20.30.101
Trudeau — Zywiec	partition 11	node 2	10.20.30.102

= the knowledge of which partition is assigned to which node

Vertical partitioning

- Goal to **reduce the I/O and performance costs** when fetching items that are frequently accessed.

Key	Name	Description	Stock	Price	LastOrdered
ARC1	Arc welder	250 Amps	8	119.00	25-Nov-2013
BRK8	Bracket	250mm	46	5.66	18-Nov-2013
BRK9	Bracket	400mm	82	6.98	1-Jul-2013
HOS8	Hose	1/2"	27	27.50	18-Aug-2013
WGT4	Widget	Green	16	13.99	3-Feb-2013
WGT6	Widget	Purple	76	13.99	31-Mar-2013



Key	Name	Description	Price
ARC1	Arc welder	250 Amps	119.00
BRK8	Bracket	250mm	5.66
BRK9	Bracket	400mm	6.98
HOS8	Hose	1/2"	27.50
WGT4	Widget	Green	13.99
WGT6	Widget	Purple	13.99

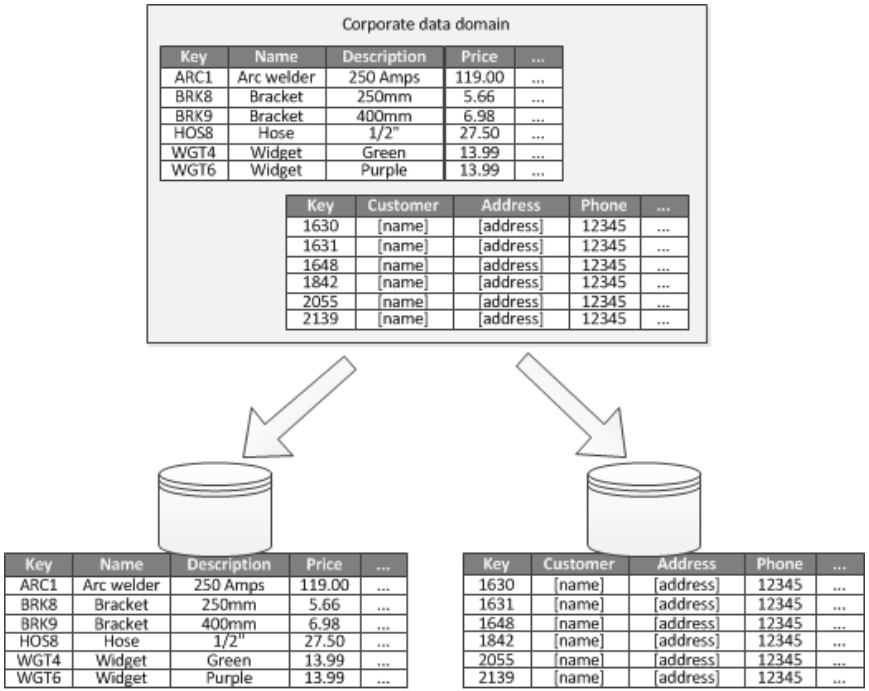
Key	Stock	LastOrdered
ARC1	8	25-Nov-2013
BRK8	46	18-Nov-2013
BRK9	82	1-Jul-2013
HOS8	27	18-Aug-2013
WGT4	16	3-Feb-2013
WGT6	76	31-Mar-2013

- Different properties of an item are stored in different partitions.
  - One partition holds data that is accessed more frequently: product name, description and price
  - Another holds inventory data: the stock count and the last ordered date.
- Application regularly gets the product name, desc. and price when displaying the product details.
- Stock count and last ordered data are commonly used together and are more frequently modified.

Vertical partitioning cont.

- **Q: Can you think of any other advantages?**
- **Other advantages:**
  - Relatively slow moving data can be separated from the more dynamic data
    - Slow moving data is a good candidate for an application to cache in memory
  - Sensitive data can be stored in a separate partition with additional security control.
- Ideally **suited** for **column-oriented data stores.**

Functional partitioning



- When possible to identify a **bounded context**, **functional partitioning** is a way to **improve isolation** and **data access performance**.
- Another common use is to separate read-write data from read-only data
- This strategy can help reduce data access contention across different parts of the system

Partitioning for scalability

- **Q: How would you approach partitioning for scalability?**

- **Analyze the application to understand the data access patterns**:
  - Result set returned by each query
  - The frequency of access
  - The inherent latency
  - The server-side compute processing requirements.

- **Determine the current and future scalability targets, such as data size and workload**
  - Distribute the data across the partitions to meet the scalability target, choose the right shard key.
  - Make sure each node has enough resources to handle the requirements in terms of storage space, processing power or network bandwidth.

- **Monitor to verify that the data is distributed well and that the partitions can handle the load**
  - Actual usage does not always match what an analysis predicts
  - It may be required to rebalance the partitions

Partitioning for query performance

- **Q: How would you approach it to improve query performance?**

- **Query performance** can be boosted by using **smaller data sets** and by running **parallel queries**.
- Each partition should contain a small proportion of the entire data set.

- Follow these steps to improve the overall query performance of your system/application.

- **Examine the application requirements and performance.**
  - **Identify** the **critical queries** that must always perform quickly.
  - **Monitor** the system to **detect any queries** that **perform slowly**.
  - **Find** which **queries** are performed **most frequently**.

- **Partition the data that causes slow performance.**

- Consider **running queries in parallel across partitions** to improve response time.

Partitioning for better availability

- **Q: How would you use partitioning to improve availability?**

- **Avoid** having the entire dataset does not constitute **a single point of failure.**

- **Consider the following factors that affect availability:**
  - **Identify critical data**
    - Consider storing critical data in highly available partitions with an appropriate back-up plan
    - Establish separate management and monitoring procedures for the different datasets
    - Place data that has the same level of criticality in the same partition
  - **Decide how to manage individual partitions**
    - If a partition fails, it can be recovered independently
    - Partition data by geographical area allows scheduled maintenance at off-peak hours
  - **Replicate critical data across partitions.**
    - This strategy can improve availability and performance, but can also introduce consistency issues related to replication lag.

We did not cover…

- **How to partition a secondary index**
  - **Document-partitioned index (local indexes)**, where the secondary index are stored in the same partition as the primary key and value.
    - Only a single partition needs to be updated on write, but a read requires scatter/gather across all.
  - **Term-partitioned index (global indexes)**, where the secondary indexes are partitioned separately, using the indexed values.
    - When a document is written, several partitions of the secondary index need to be updated; however a read can be served from a single partition.

- **Creating materialized views that summarize data to support fast query operations**.
  - Useful in a partitioned data store if the partitions that contain the data being summarized are distributed across multiple sites.

- **Parallel Query Execution in presence of partitions**
- **Distributed Transactions** (later in class)

Summary

- **Partitioning is necessary when data and load volume exceeds a single machine's capacity.**
- The goal is to **spread the data and query load evenly across multiple machines**, avoiding hotspots.
- Need to be careful when choosing the partitioning scheme so that it is appropriate to the data and workload properties, and rebalance it when nodes are added/removed.
- **Three main types of partitioning**:
  - horizontal,
  - vertical and
  - functional.
- **Two main approaches for horizontal partitioning: key range and hash-based.**
- **Various techniques for rebalancing and routing.**

References

The material covered in this class is mainly based on:

- The book *“Designing Data-Intensive Applications – The Big Ideas Behind Reliable, Scalable, and Maintainable Systems”* by Martin Kleppmann (Chapters 5 and 6) (link)

Some information and images were based on material from:

- Microsoft’s Azure Application Architecture Guide
  - Best practices for horizontal, vertical and functional data partitioning (link)
  - Data partitioning strategies in various Azure services (link)
  - Sharding pattern (link)